DETAILED ACTION

1. Claims 1-8, 10-22, 24-30, 33-38, 40, 41, 43, 46-50, 52-56, 59-64 and 67-73 are allowed.

Examiner's Amendment

2. An examiner's amdment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

3. A telephone interview with George S. Bardmesser (Reg. No 44,020) on 8/28/2008 discussed the allowable subject matter of the claimed invention. Authorization for the examiner's amendemnt was given by Mr. Bardmesser during the telephone interview. Examiner's amendment is necessitated to further clarify the claimed invention.

4. Claims 1, 9, 10, 27, 31-34, 40, 44-47, 53, 57, 59, 60, and 67-71 have been further amended below.

1. (Currently Amended) A method of on-the-fly patching of executable code comprising:

identifying original instructions to be changed while the original instructions are being executed on a processor;

copying the original instructions to a storage location;

adding a hook with a first jump instruction for transferring control to the copied instructions;

using a second jump instruction in the copied instructions for transferring control to an unpatched instruction in a location where the instructions are being patched; and

~~using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all,~~ replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions **by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all**; and

calling the hook,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called;

wherein the original instructions are part of the instruction set of the processor available to a user, ~~and~~

wherein a number of times the mark instructions have been executed is counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions**, and**

wherein the modified instructions determine the number of the instructions at the location of the original code that had already been executed.

9. (Cancelled)

10. (Currently Amended) The method of claim **[[9]] 1**, wherein the **modified instructions include a** resolver **that** determines a number of instructions that had already been executed using the mark instructions.

27. (Currently Amended) A method of on-the-fly patching of executable code comprising:

verifying that original instructions to be patched are susceptible to patching while the original instructions are being executed on a processor;

generating pseudooriginal code by copying the original instructions to a different storage location from the original instructions;

adding a hook with a first jump instruction to the pseudoorignal code transferring control to the pseudoorinal instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched;

~~using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all,~~ replacing the original code while the original instructions are in the process of being executed on the processor with tag instructions that indicate only their execution **by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all**; and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called,

wherein the tag instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, ~~and~~

wherein a number of times the tag instructions have been executed is counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions**, and**

wherein the modified instructions determine the number of the instructions at the location of the original code that had already been executed.

31. (Cancelled)

32. (Cancelled)

33. (Currently Amended) The method of claim **[[32]]** **27**, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be patched, ~~**the resolver calls**~~ the pseudooriginal code **is called** so as to imitate a "no patch installed" scenario.

34. (Currently Amended) The method of claim 33, wherein, after execution of the pseudooriginal code, ~~**the resolver returns**~~ control **is returned** to the next instruction.

40. (Currently Amended) A method of on-the-fly patching of executable code comprising:

identifying original instructions to be patched while the original instructions are being executed on a processor;

allocating a storage location for storing a functionally equivalent copy of the original instructions;

copying the original instructions to the storage location;

adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched;

~~using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all,~~ replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions **by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all**; and

calling he hook,

wherein the copied instructions include a first jump instruction to the pseudooriginal code and the hook includes a second jump instruction that returns operations to patched instructions after the original instructions,

wherein the hook activates the second jump instruction to transfer control to the pseudoorignal instructions at a location just after the original instruction from which the hook was called,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched,

wherein the original instructions are part of the instruction set of the processor available to a user, ~~and~~

wherein a number of times the mark instructions have been executed is counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions**, and**

wherein the modified instructions determine the number of the instructions at the location of the original code that had already been executed.

44. (Cancelled)

45. (Cancelled)

46. (Currently Amended) The method of claim **[[45]]** **40**, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be patched, ~~**the resolver calls**~~ the functionally equivalent copy **is called** so as to imitate a "no patch installed" scenario.

47. (Currently Amended) The method of claim 46, wherein, after execution of the functionally equivalent copy, ~~**the resolver returns**~~ control **is returned** to the next instruction.

53. (Currently Amended) A computer useable removable storage unit having computer program logic stored thereon for executing on a processor, for on-the-fly patching of executable code, the computer program logic comprising:

computer program code means for identifying original instructions to be patched while the original instructions are being executed on a processor;

computer program code means for copying the original instructions to a storage location;

computer program code means for adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched;

~~using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all,~~ computer program code means for replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions **by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all**; and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, ~~and~~

wherein a number of times the mark instructions have been executed is counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions**, and**

wherein the modified instructions determine the number of the instructions at the location of the original code that had already been executed.

57. (Cancelled)

59. (Currently Amended) The computer program logic of claim 53, wherein, if the number of instructions that had already been executed is less than a number of original instructions to be patched, ~~**the resolver calls**~~ the copied instructions at the storage location **are called** so as to imitate a "no patch installed" scenario.

60. (Currently Amended) The computer program logic of claim 59, wherein, after execution of the instructions to the storage location, ~~**the resolver returns**~~ control **is returned** to the next instruction.

67. (Currently Amended) A computer useable removable storage unit having computer program logic stored thereon for executing on a processor, for on-the-fly patching of executable code, the computer program logic comprising:

computer program code means for verifying that original instructions to be patched are susceptible to patching while the original instructions are being executed on a processor;

computer program code means for generating pseudooriginal code from the original instructions at a different storage location from the original instructions;

computer program code means for adding a hook to the pseudooriginal code with a first jump instruction transferring control to the pseudoorignal instructions by using a second jump instruction in the pseudooriginal instructions that always transfers control to unpatched instructions in a location where the instructions are being patched;

~~using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all,~~ computer program code means for replacing the original code while the original code is in the process of being executed on the processor with tag instructions that indicate only their execution **by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all**; and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the pseudoorignal instructions at a location just after the original instruction from which the hook was called,

wherein the tar instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, ~~and~~

wherein a number of times the tag instructions have been executed is counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions**, and**

wherein the modified instructions determine the number of the instructions at the location of the original code that had already been executed.

68. (Currently Amended) A computer useable removable storage unit having computer program logic stored thereon for executing on a processor, for on-the-fly patching of executable code, the computer program logic comprising:

computer program code means for identifying original instructions to be patched while the original instructions are executed on a processor;

computer program code means for allocating a storage location for storing a functionally equivalent copy of the original instructions;

computer program code means for copying the original instructions to the storage location;

computer program code means for adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched;

~~**computer program code means for using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all,**~~ computer program code means for replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions **by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all**; and

computer program code means for calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, ~~and~~

wherein a number of times the mark instructions have been executed is counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions**, and**

wherein the modified instructions determine the number of the instructions at the location of the original code that had already been executed.

69. (Currently Amended) A system for on-the-fly patching of executable code comprising:

a processor;

a memory operatively coupled to the processor;

computer code loaded into the memory for implementing the functionality of:

means for identifying original instructions to be patched while the instructions are being executed on **the [[a]]** processor;

means for copying the original instructions to a storage location;

means for adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched;

~~**using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all,**~~ means for replacing the original code as while the code is in the process of being executed on the processor with mark instructions **by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all;** and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called,

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched, ~~**and**~~

wherein a number of times the mark instructions have been executed is counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions**, and**

wherein the modified instructions determine the number of the instructions at the location of the original code that had already been executed.

70. (Currently Amended) A system for on-the-fly patching of executable code comprising:

a processor;

a memory operatively coupled to the processor;

computer code loaded into the memory for implementing the functionality of:

means for verifying that original instructions to be patched are susceptible to patching while the instructions are being executed on **the [[a]]** processor;

means for generating pseudooriginal code from the original instructions at a different storage location from the original instructions;

means for adding a hook with a first jump instruction to the pseudooriginal instructions transferring control to the pseudoorignal instructions by using a second jump instruction in the pseudoorignal instructions that always transfers control to unpatched instructions in a location where the instructions are being patched;

~~using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all,~~ means for replacing the original code while the original code is in the process of being executed on the processor with tag instructions that indicate only their execution **by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all;** and

calling the hook,

wherein the original instructions are part of the instruction set of the processor available to a user,

wherein the tag instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched;

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called, ~~and~~

wherein a number of times the tag instructions have been executed is counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions**, and**

wherein the modified instructions determine the number of the instructions at the location of the original code that had already been executed.

71. (Currently Amended) A system for on-the-fly patching of executable code comprising:

a processor;

a memory operatively coupled to the processor;

computer code loaded into the memory for implementing the functionality of:

means for identifying original instructions to be changed while the instructions are executed on **the [[a]]** processor;

means for allocating a storage location for storing a functionally equivalent copy of the original instructions;

means for copying the original instructions to the storage location;

means for adding a hook with a first jump instruction transferring control to the copied instructions by using a second jump instruction in the copied instructions that always transfers control to unpatched instructions in a location where the instructions are being patched;

~~using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all,~~ means for replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions **by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all**; and

calling the hook;

wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched,

wherein the hook activates the second jump instruction to transfer control to the copied instructions at a location just after the original instruction from which the hook was called**,**

wherein the original instructions are part of the instruction set of the processor available to a user, ~~**and**~~

wherein a number of times the mark instructions have been executed is counted to determine a location of the processor's instruction pointer where execution should resume, in the patched instructions**, and**

wherein the modified instructions determine the number of the instructions at the location of the original code that had already been executed.

Examiner's Statement of Reason(s) for Allowance

5. The following is an examiner's statement of reasons for allowance:

The prior art of record (Mulchandani et al. USPN 6,112,025), taken alone or in combination with other fails to teach or reasonably suggest in combination with other claimed limitations *replacing the original instructions while the original instructions are in the process of being executed on the processor with mark instructions (applicants*

indicate, for the record that neither NOP, CALL, nor RET instructions can be used as the claimed mark/tag instructions) by using atomic writes that guarantee that a result of the operation can be observed as completed or not observed at all; wherein the mark instructions place an identifiable set of data into a processor stack that is identified at a later time, such that the stack contains a number of how many instructions have been patched; wherein a number of times the mark instructions have been executed is counted to determine a location of the processor's instruction pointer where executed should resume, in the patched instructions as recited in claim 1. Similar concepts are found in other independent claims.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Phillip H. Nguyen whose telephone number is (571) 270-1070. The examiner can normally be reached on Monday - Thursday 10:00 AM - 3:00 PM EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Y. Zhen can be reached on (571) 272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

PN
8/29/2008

/Wei Y Zhen/
Supervisory Patent Examiner, Art Unit 2191